**得分：**

**东莞城市学院**

**课程作业**

**课程： 数据结构**

**章节： 第2章 线性表**

**学院： 计算机与信息学院**

**班级： 2020级 华为创新班**

**学号： 202035020145**

**姓名： 张宇航**

**教师： 彭 刚**

**2021年 9 月 23 日**

习 题

1．选择题

（1）顺序表中第一个元素的存储地址是100，每个元素的长度为2，则第5个元素的地址是（ B ）。

A．110 B．108 C．100 D．120

（2）在n个结点的顺序表中，算法的时间复杂度是O(1)的操作是（ A ）。

A．访问第i个结点（1≤i≤n）和求第i个结点的直接前驱（2≤i≤n）

B．在第i个结点后插入一个新结点（1≤i≤n）

C．删除第i个结点（1≤i≤n）

D．将n个结点从小到大排序

（3） 向一个有127个元素的顺序表中插入一个新元素并保持原来顺序不变，平均要移动 的元素个数为（ B ）。

A．8 B．63.5 C．63 D．7

（4）链接存储的存储结构所占存储空间（ A ）。

A．分两部分，一部分存放结点值，另一部分存放表示结点间关系的指针

B．只有一部分，存放结点值

C．只有一部分，存储表示结点间关系的指针

D．分两部分，一部分存放结点值，另一部分存放结点所占单元数

（5）线性表若采用链式存储结构时，要求内存中可用存储单元的地址（ D ）。

A．必须是连续的 B．部分地址必须是连续的

C．一定是不连续的 D．连续或不连续都可以

（6）线性表Ｌ在（ B ）情况下适用于使用链式结构实现。

A．需经常修改Ｌ中的结点值 Ｂ．需不断对Ｌ进行删除插入

C．Ｌ中含有大量的结点 Ｄ．Ｌ中结点结构复杂

（7）单链表的存储密度（ C ）。

A．大于1 B．等于1 C．小于1 D．不能确定

（8）将两个各有n个元素的有序表归并成一个有序表，其最少的比较次数是（ A ）。

A．n B．2n-1 C．2n D．n-1

（9）在一个长度为n的顺序表中，在第i个元素（1≤i≤n+1）之前插入一个新元素时须向后移动（ B ）个元素。

A．n-i B．n-i+1 C．n-i-1 D．I

(10) 线性表L=(a1，a2,……an)，下列说法正确的是（ D ）。

A．每个元素都有一个直接前驱和一个直接后继

B．线性表中至少有一个元素

C．表中诸元素的排列必须是由小到大或由大到小

D．除第一个和最后一个元素外，其余每个元素都有一个且仅有一个直接前驱和直接后继。

(11) 创建一个包括*n*个结点的有序单链表的时间复杂度是（ C ）。

A．O(1) B．O(n) C．O(n2) D．O(nlog2n)

(12) 以下说法错误的是（ D ）。

A．求表长、定位这两种运算在采用顺序存储结构时实现的效率不比采用链式存储结构时实现的效率低

B．顺序存储的线性表可以随机存取

C．由于顺序存储要求连续的存储区域，所以在存储管理上不够灵活

D．线性表的链式存储结构优于顺序存储结构

(13) 在单链表中，要将s所指结点插入到p所指结点之后，其语句应为（ D ）。

A．s->next=p+1; p->next=s;

B．(\*p).next=s; (\*s).next=(\*p).next;

C．s->next=p->next; p->next=s->next;

D．s->next=p->next; p->next=s;

(14) 在双向链表存储结构中，删除p所指的结点时须修改指针（ A ）。

A．p->next->prior=p->prior; p->prior->next=p->next;

B．p->next=p->next->next; p->next->prior=p;

C．p->prior->next=p; p->prior=p->prior->prior;

D．p->prior=p->next->next; p->next=p->prior->prior;

(15) 在双向循环链表中，在p指针所指的结点后插入q所指向的新结点，其修改指针的操作是（ C ）。

A．p->next=q; q->prior=p; p->next->prior=q; q->next=q;

B．p->next=q; p->next->prior=q; q->prior=p; q->next=p->next;

C．q->prior=p; q->next=p->next; p->next->prior=q; p->next=q;

D．q->prior=p; q->next=p->next; p->next=q; p->next->prior=q;

答案：BABADBCABDCDDAC

2．算法设计题

（1）将两个递增的有序链表合并为一个递增的有序链表。要求结果链表仍使用原来两个链表的存储空间, 不另外占用其它的存储空间。表中不允许有重复的数据。

*#include* <stdio.h>

*#include* <stdlib.h>

*typedef* *int* ElementType;

*typedef* *struct* LNode{

    ElementType elem;

*struct* LNode*\** next;

}LNode,*\**LinkList;

*void* MergeOrderList (LinkList *&*La, LinkList *&*Lb, LinkList *&*Lc){

    LNode\* p = La->next;

    LNode\* q = Lb->next;

    LNode\* r;

    Lc = La;

    Lc->next = *NULL*;

    free(Lb);

    r = Lc;

*while*(p != *NULL* && q != *NULL*){

*if*(p->elem <= q->elem){

            r->next = p;

            p = p->next;

            r = r->next;

        }*else*{

            r->next = q;

            q = q->next;

            r = r->next;

        }

    }

    r->next = *NULL*;

*if*(p != *NULL*){

        r->next = p;

    }

*if*(q != *NULL*){

        r->next = q;

    }

}

（5）设计算法将一个带头结点的单链表A分解为两个具有相同结构的链表B、C，其中B表的结点为A表中值小于零的结点，而C表的结点为A表中值大于零的结点（链表A中的元素为非零整数，要求B、C表利用A表的结点）。

*#include* <stdio.h>

*#include* <stdlib.h>

*typedef* *int* ElementType;

*typedef* *struct* LNode{

    ElementType elem;

*struct* LNode*\** next;

}LNode,*\**LinkList;

*void* resolve(LinkList *&*main,LinkList *&*master){

    LinkList pre = main;

    LinkList p\_main = main;

    LinkList p\_master = master;

    p\_main = p\_main->next;

*while*(p\_main){

*if*(p\_main->elem < 0){

            LinkList p = *new* LinkList;

            p = p\_main;

            pre->next = p\_main->next;

            p\_main = p\_main->next;

            p->next = *NULL*;

            p\_master->next = p;

            p\_master = p;

        }*else*{

            pre = p\_main;

            p\_main = p\_main->next;

        }

    }

}

（7）设计一个算法，通过遍历一趟，将链表中所有结点的链接方向逆转，仍利用原表的存储空间。

*#include* <stdio.h>

*#include* <stdlib.h>

*typedef* *int* ElementType;

*typedef* *struct* LNode{

    ElementType elem;

*struct* LNode*\** next;

}*\**LinkList;

*void* reverse\_linklist(LinkList*&* L);

*void* reverse\_linklist(LinkList*&* L){

    LinkList q,p = L->next;

    L->next = *NULL*;

*while*(p){

        q = p->next;

        p->next = q->next;

        L->next = p;

        p = q;

    }

}